

Contenido

INTRODUCCIÓN	138
ESCRITURA DE UN ARCHIVO XML	138
LECTURA DE UN ARCHIVO XML	141

8.

XML

INTRODUCCIÓN

Los archivos *xml* no son utilizados únicamente para definir la parte gráfica de nuestro programa en *Android*. También podemos usarlo para almacenar información de forma permanente, ya sea en local o en un servidor.

Evidentemente existen formas más eficientes para almacenar información de forma más cómoda y sencilla, sobre todo cuando esta información es compleja. Por ejemplo si al crear una Base de Datos, tienes una sola tabla, quizás deberías usar *xml* en su lugar.

Existen diferentes formas de trabajar con archivos *xml*: *DOM*, *SAX* y *XmlPull*.

Las aplicaciones de Android, en su mayor parte, están programadas en *Java* y esto significa que podríamos usar las mismas estrategias naturales de este lenguaje para realizar nuestras lecturas y escrituras de ficheros XML (*DOM* y *SAX*).

Sin embargo *Android* no es sólo *Java*, sino que es un *Sistema Operativo* para dispositivos móviles. Esto significa que, no deberíamos usar un *DOM parser* para leer un fichero XML pues en ciertos casos podemos quedarnos sin memoria (ficheros muy grandes). Para estas tareas *Android* va a usar unas interfaces similar al *SAX parser*, llamadas [XmlSerializer](#) (escritura) y [XmlPullParser](#) (lectura).

ESCRITURA DE UN ARCHIVO XML

Lo importante del funcionamiento de la interfaz *XmlSerializer* es que nosotros seremos responsables de decir que empezamos una etiqueta, que creamos atributos, que ponemos texto y que cerramos la etiqueta. Su filosofía es más cercana al *SAX parser* que al *DOM parser*: mientras que *DOM* crea todo el árbol en la memoria y podemos acceder a cada nodo (*cada etiqueta*) para manipularla, en este caso sólo podemos hacer cosas con la última etiqueta que hemos dejado abierta. Dicho de otra forma, *no hay navegación*. La ventaja evidente es que necesita menos memoria que *DOM*.

Lo primero que debemos hacer es crear un fichero en el cual vayamos a escribir. Debemos recordar Android es básicamente un sistema operativo Linux y, en este caso, esto significa que cada *.apk* que instalamos recibe su propio *User-ID*. Además, cuando instalamos una aplicación, se crea un directorio propio del paquete en el directorio */data/data* de la memoria interna al que sólo puede acceder este *User-ID* y el superusuario *root*. Podemos, por tanto, crear tres tipos de fichero:

- *MODE_PRIVATE*: sólo la propia aplicación (y *root*) puede acceder para leer y escribir.



- `MODE_WORLD_READABLE`: cualquier *User-ID* puede leer el fichero.
- `MODE_WORLD_WRITEABLE`: cualquier *User-ID* puede escribir en el fichero.

Para nuestro ejemplo vamos a suponer que lo que queremos generar es un fichero XML que contenga información del estado de la aplicación (podría ser una configuración de las opciones, podrían ser las stats de un jugador si nuestra aplicación fuera un juego...), de modo que usaremos `MODE_PRIVATE`.

El archivo xml que queremos generar sería algo así:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<lista_puntuaciones>
  <puntuaciones fecha="xxx">
    <nombre>Pepe Balmaseda del Alamo</nombre>
    <puntos>35000</puntos>
  </puntuaciones>
  <puntuaciones fecha="yyy">
    <nombre>Juan Pérez Sánchez</nombre>
    <puntos>30000</puntos>
  </puntuaciones>
</lista_puntuaciones>
```

Empecemos:

```
private void escribirXML() {
    FileOutputStream fout = null;
    try {
        fout = openFileOutput("test.xml", MODE_PRIVATE);
    } catch (FileNotFoundException e) {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
    }
}
```

El método `openFileOutput()` recibe el nombre del fichero que va a escribir y el modo de éste (de entre los tres mencionados). Lo siguiente es crear el serializador de XML, el cual recibe un `FileOutputStream`. Además vamos a ponerle que la codificación sea UTF-8 y que se indente correctamente. Para ello agregamos el código que viene a continuación antes del final del código superior:

```
XmlSerializer serializer = Xml.newSerializer();
try {
    serializer.setOutput(fout, "UTF-8");
    serializer.startDocument(null, true);
    serializer.setFeature("http://xmlpull.org/v1/doc/features.html#indent-output", true);
} catch (Exception e) {
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}
```

Con `setOutput()` elegimos el fichero donde escribiremos y la codificación de caracteres. Con `startDocument()` decimos que comience a crear el árbol del XML en memoria, el atributo booleano indica si el fichero es *standalone* (*independiente*) o no. Y la característica que



ponemos con `setFeature()` en este caso es que se indente para poder visualizarlo más fácilmente.

Sobre el objeto `XmlSerializer` tenemos cuatro funciones:

- `startTag()`: recibe el *Namespace* (en este caso *null* porque no hay *Namespace*) y el nombre de la etiqueta que va a crear.
- `endTag()`: recibe los mismos parámetros que el anterior y sirve para cerrar una etiqueta abierta previamente con ese nombre. En realidad el serializador sólo genera una etiqueta de cierre con dicho nombre, se haya abierto antes algo del mismo nombre o no. Hay que tener cuidado al usarlo, por tanto. ¡Pero que no falte!
- `attribute()`: recibe el *Namespace*, el nombre del atributo y el valor del atributo. Todos los componentes son *Strings*, y hay que tenerlo en cuenta si sacamos dichos valores de variables (como enteros, por ejemplo).
- `text()`: recibe el texto que se va a poner entre el comienzo y el fin de una etiqueta.

Hay que colocar el código siguiente antes del `catch`:

```
//Empizo a defini xml
serializer.startTag(null, "lista_puntuaciones");

//Un elemento
serializer.startTag(null, "puntuaciones");
serializer.attribute(null, "fecha", "xxx");
serializer.startTag(null, "nombre");
serializer.text("Pepe Balmaseda del Alamo");
serializer.endTag(null, "nombre");
serializer.startTag(null, "puntos");
serializer.text("35000");
serializer.endTag(null, "puntos");
serializer.endTag(null, "puntuaciones");

//Otro elemento
serializer.startTag(null, "puntuaciones");
serializer.attribute(null, "fecha", "yyy");
serializer.startTag(null, "nombre");
serializer.text("Juan Pérez Sánchez");
serializer.endTag(null, "nombre");
serializer.startTag(null, "puntos");
serializer.text("30000");
serializer.endTag(null, "puntos");
serializer.endTag(null, "puntuaciones");

//Termino xml
serializer.endTag(null, "lista_puntuaciones");

serializer.endDocument();
serializer.flush();
fout.close();
```

Si ejecutamos ahora nuestra aplicación y abrimos un explorador de ficheros con permisos de superusuario (para poder navegar hasta el directorio) como *RootExplorer*, tras navegar al directorio `/data/data/miaplicacion/files/test.xml`, podemos abrirlo y ver el contenido.



LECTURA DE UN ARCHIVO XML

Una vez que hemos ejecutado nuestra aplicación con la escritura del fichero tendremos un ficherito *test.xml* en el directorio privado de la aplicación. El siguiente paso va a ser leerlo y volcarlo tal cual. Para no complicarnos las cosas, y como esta aplicación es sólo para fines didácticos (para uso con Eclipse directamente), todo lo que leamos del fichero lo vamos a guardar en un *arrayList* y lo visualizaremos en un *listView* que pondremos en nuestra *activity*.

Lo siguiente será crear un método en el que vayamos a leer. El contenido va a ser análogo al método *escribirXML()*. Los pasos serán: abrir el fichero del que vamos a leer, usar un *XmlPullParser* para ir recorriendo el XML e ir almacenando los elementos en el *arrayList*.

Previamente habremos definido una clase asociada a los datos que queremos almacenar en el *arrayList*, y asociamos el fichero que queremos leer a la variable *FileInputStream*.

El siguiente caso será vincular un objeto *XmlPullParser* a dicho fichero e iniciamos la variable evento y la variable auxiliar. Evento nos permitirá ir reconociendo que tipo de elemento *Xml* estamos tratando (*START_DOCUMENT*, *START_TAG*, *END_TAG* o *TXT*), de forma que vayamos tomando decisiones según proceda. En aux almacenaremos los datos concretos obtenidos del fichero *Xml*.

```
private void leerXML(){

    XmlPullParser parser = Xml.newPullParser();
    FileInputStream fin = null;

    try {
        fin=openFileInput("test.xml");
        parser.setInput(fin, null);

        //Evento que voy leyendo
        int evento = parser.getEventType();

        //Objeto del tipo datos
        Datos aux = null;

        //lista donde guardamos los datos leídos
        lista= null;
    }
```

Para realizar la lectura entran en juego un par de conceptos. En el caso del *XmlPullParser* tenemos que ir avanzando usando el método *next()* cada vez que queramos el siguiente elemento. Dicho elemento viene dado por un evento, que puede ser: comienzo de documento *START_DOCUMENT*, apertura de etiqueta *START_TAG*, cierre de etiqueta *END_TAG*, comienzo de texto y fin de documento *END_DOCUMENT*. Para leer el fichero vamos a ir evento por evento hasta encontrar el fin del documento. Evidentemente, es necesario conocer la estructura del archivo *Xml* que estamos procesando. Así cuando encontremos la etiqueta de comienzo de documento, en nuestro caso debemos crear la lista. Cuando encontremos una etiqueta de *START_TAG*, debemos saber si es una etiqueta del tipo “*puntuaciones*” o bien si es del tipo “*nombre*” o “*puntos*”. Si es de tipo “*puntuaciones*”, debemos crear un nuevo objeto de tipo datos, donde posteriormente ir almacenando lo que vayamos leyendo. Además debemos recuperar el atributo “*fecha*” de esta etiqueta, esto lo hacemos con *parser.getAttributeValue(null, "fecha")*. Cuando finalicemos la lectura de la etiqueta






“puntuaciones”, habrá que añadir el elemento leído a la lista. Cuando se termine la lectura del documento tendremos que crear el adaptador y visualizar los datos en la lista.

Veamos el código:

```
//Bucle para recorrer el archivo XML
while(evento!=XmlPullParser.END_DOCUMENT){
    switch (evento){
        case XmlPullParser.START_DOCUMENT:
            lista=new ArrayList<Datos>();
            break;
        case XmlPullParser.START_TAG:
            //Si es la etiqueta puntuaci-n creamos un objeto de tipo datos y recuperamos la fecha
            if(parser.getName().equals("puntuaciones")){
                aux=new Datos();
                aux.setFecha(parser.getAttributeValue(null, "fecha"));
            }
            else
                if(parser.getName().equals("nombre")){
                    //guardamos el nombre
                    aux.setNombre(parser.nextText());
                }
                else if(parser.getName().equals("puntos")){
                    //Guardamos puntuaci-n
                    aux.setPuntos(parser.nextText());
                }
            break;
        case XmlPullParser.END_TAG:
            //Si hemos terminado de leer un elemento completo del archivo Xml, lo a-adimos a la lista
            if(parser.getName().equals("puntuaciones")){
                lista.add(aux);
            }
            break;
    }
    //avanzamos en la lectura
    evento=parser.next();
}
fin.close();

Toast.makeText(this, "Leido correctamente", Toast.LENGTH_LONG).show();

} catch (Exception e) {
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}
AdaptadorDatos adaptador = new AdaptadorDatos(this);
milista.setAdapter(adaptador);
}
```

-  **Ejercicio Propuesto LeerEscribirXML**
-  **Ejercicio Propuesto LeerPuntuacionesInternet**
-  **Ejercicio Propuesto PelículasEnCartelera**

